RESORT for Java Toolset:
S/W 품질 도구 소개 및
S/W 감리 적용 사례

Soft4Soft
http://www.soft4soft.com

Part 1: S/W 품질 도구 소개

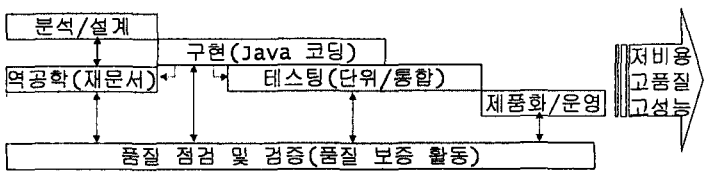
- 회사 소개
- RESORT for Java 제품 소개
- RESORT for Java 제품 기능
  - RESORT for Java Reverse Engineering 주요 기능
  - RESORT for Java Quality 주요 기능
  - RESORT for Java Testing 주요 기능
- 기대 효과

회사 소개

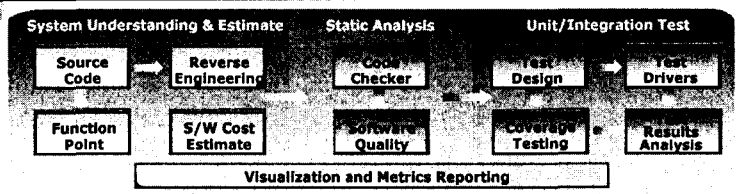
- 2001년 법인 설립(ETRI 벤처 창업 기업)
- 주사업 분야
  - 소프트웨어 품질 솔루션 도구 개발(Java, C, C++, C#)
  - 소프트웨어 품질 컨설팅 및 교육
- 제품
  - RESORT for C
  - RESORT for Java, JavaFP
  - RESORT for C++, C++FP – 출시 예정
  - RESORT for C#, C#FP – 출시 예정
- 기술 인증
  - IT 마크 인증 – 정통부
  - GS 품질 인증 – 정통부
  - KT 마크 인증 – 과기부
  - 삼성 SDS S/W 협력회사
- 품질 컨설팅(감리 수행)
  - 대법원, 한국은행
  - 육군 C4I
  - CBS(인터넷수능시스템)

RESORT 제품 소개

- 품질관리 자동화솔루션
  - 구현 초기부터 통합 테스팅까지의 개발 과정에서 소프트웨어 개발 및 유지 보수의 품질 향상을 위한 품질 프로세스 제공
    - 소프트웨어 에러를 사전에 예방 및 S/W 품질 향상
    - S/W 특성(설계/구현/테스트)을 분석 및 평가
    - 테스트 계획의 시간과 노력 절감
    - Project 관리 용이



RESORT 제품 기능



도구 종류	쓰 임 새	기능 종류
역공학	구현 중 또는 운영중인 S/W의 구조 및 이해 요구사항 및 설계준수 여부 검토 S/W구조상의 문제점 조기파악 및 대처방안 지원	Class/Package Sequence/Collaboration Control-Flow/Data-Flow Source Code Browser
품질 보증	Code Checker를 통한 코드지침 준수 검토 S/W Metrics를 통한 코드 품질 진단 품질개선을 위한 효율인 방안 제시	Code Checker OO Metrics Halstead/Package Metrics Quality/System Metrics
테스트	효율적인 시나리오 작성 지원 테스트 우선순위 제공 및 중복 테스트 제거 테스트 결과를 통하여 병목현상 및 메모리 누수 파악	Design Test Case Control-Flow Testing Data-Flow Testing Sequence Testing
기능 점수	기능 점수 및 소프트웨어 개발비 산정 소프트웨어 생산성 및 비용 평가	FP Size FP Estimator FP Counter

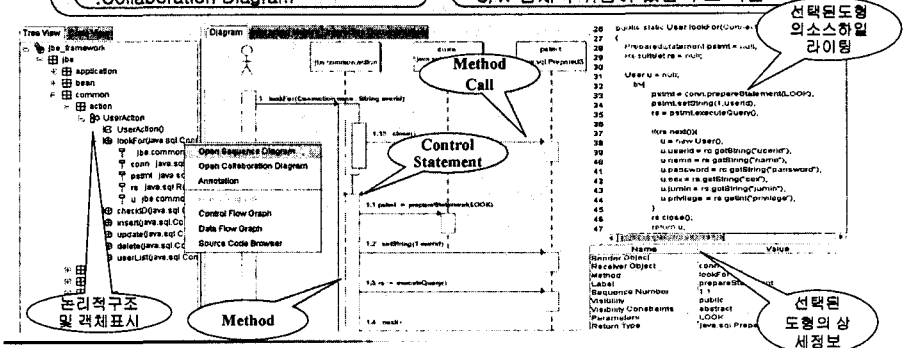
Soft4Soft

RESORT 역공학 기능(Diagram)

Reverse Engineering – Diagram 자동 생성

-완벽한 UML 1.4 Diagram 자동 생성
.Class Diagram
.Sequence Diagram
.Collaboration Diagram

-S/W 상세 설계 및 아키텍처 분석 및 이해
-S/W 산출물간의 정확(일치)성 확인
-구현레벨에서의 요구사항 반영확인
-S/W 잠재적 위험이 있는 구조 식별



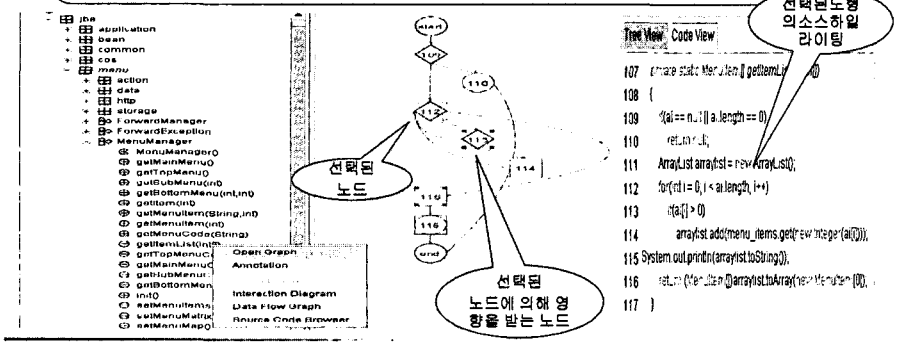
Soft4Soft

RESORT 역공학 기능(Graph)

Reverse Engineering – Graph 자동 생성

- 완벽한 Procedural Graph 자동 생성
  .Data-Flow Graph
  .Control-Flow Graph

-함수내의 구체적인 컨트롤의 흐름 파악
-복잡한 구조(문제 발생가능성 큼)에 대한 검토(복잡도, 제어 구조 깊이)



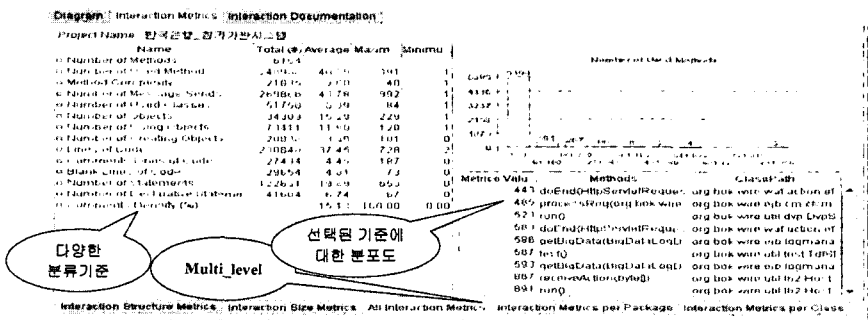
Soft4Soft

7

RESORT 역공학 기능(Report)

Reverse Engineering – 품질 분석 보고서(100+ Metrics)

-Class Diagram: 라인 크기, 주석 비율, 복잡도, 결합력, 응집력, 상속성, 재사용 비율 등
-Sequence Diagram: 메소드 호출, 메시지, 객체 사용 등등
-Control Flow Graph: 복잡도, 중첩 제어구조, Branch/Exception/Jump/Out 문장 등등
-Data Flow Graph: 전역 변수, 지역 변수, 사용/정의 변수 등등



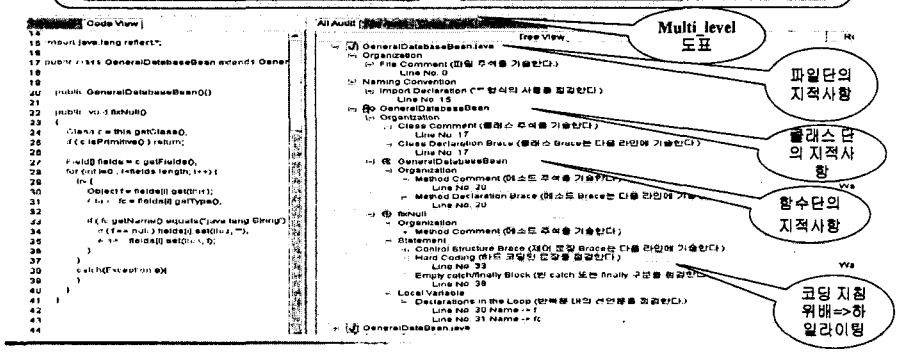
Soft4Soft

8

RESORT 품질보증기능(Inspection)

Quality Assurance – Code Checker(Inspection 기능)

- -코딩 지침 설정 (File, Class, Method)
  : 120+ (Guideline, Rule)
- -코딩 지침 준수 여부를 검사
- -코드 결함과 Code 간의 매핑
- -코드 이해성 및 판독성 검사
  : S/W 유지보수 용이
- -코드 성능성 및 메모리 누수 검사
- -코딩 표준화 및 프로그램 개발 능력 향상



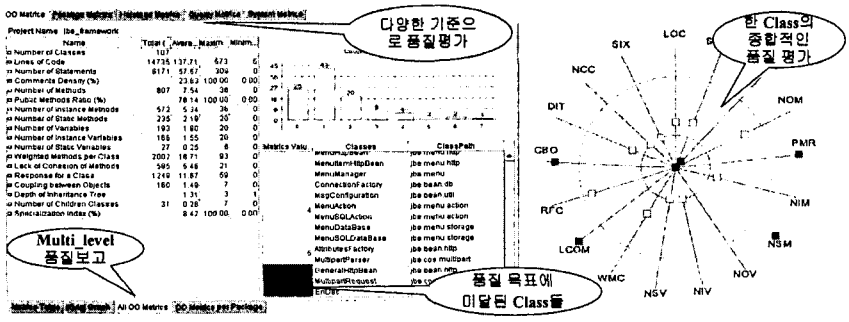
Soft4Soft

9

RESORT 품질보증기능(Metrics)

Quality Assurance – Software Metrics

- -OO Metrics 기준 값 설정
  : 80+(Kiviat, Bar, Circle Graph 생성)
- -최적화 코드 검증
- -품질 개선을 위해 고 위험 Class 식별
- -비 최적화된 코드 평가
  : 프로그램 크기 축소 및 실행 시간 단축
- -S/W 고 위험성 및 복잡성 품질 평가
- -S/W 판독성 및 유지보수성 품질 평가



Soft4Soft

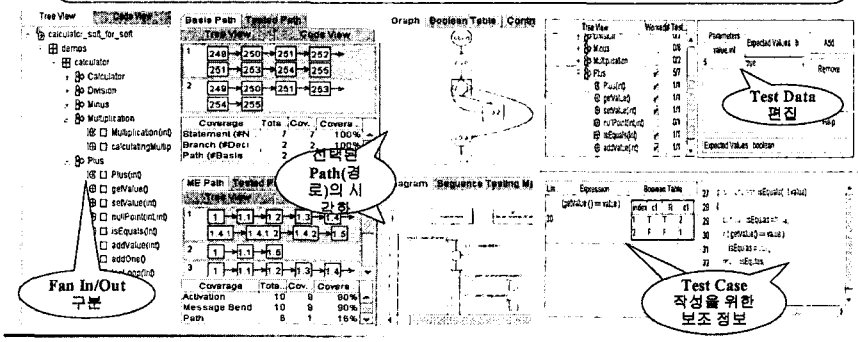
10

RESORT 테스트 기능(Plan&Design)

Testing – 계획 및 설계 작성

Test Plan
- -최소 경로 (Basis Path==V(g))
- -테스트 우선순위제공,Bolean Table 제공
- -메시지 흐름 경로(ME-Path)

Test Case Design
- -Test Case/Scenario 설계 UI 제공(Test Driver)
- -Code Instrumentation 기능제공(Web)



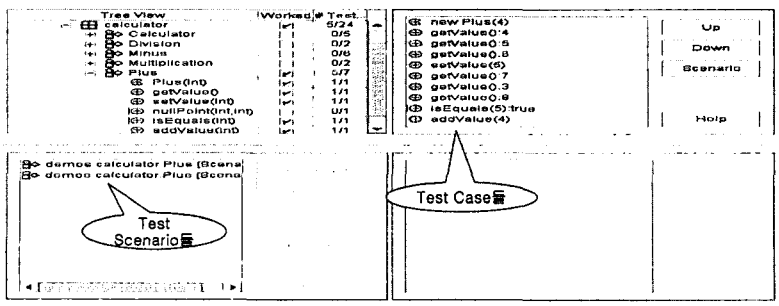
Soft4Soft

11

RESORT 테스트 기능(Design)

Testing – 계획 및 설계 작성(계속)

Test Scenario 설계 UI 지원
- Class, Package 또는 Project(system) 별 시나리오 작성
- Test Suite 작성
- JUnit Framework 자동 변환 기능 제공



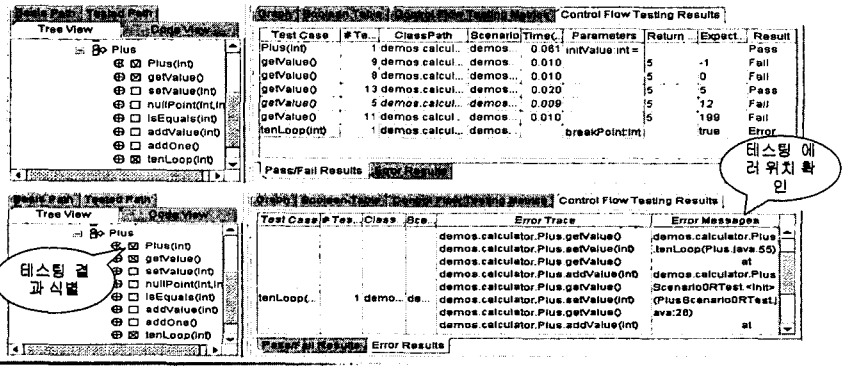
Soft4Soft

12

RESORT 테스트 기능(Evaluation)

Unit/Integration Testing 결과 분석

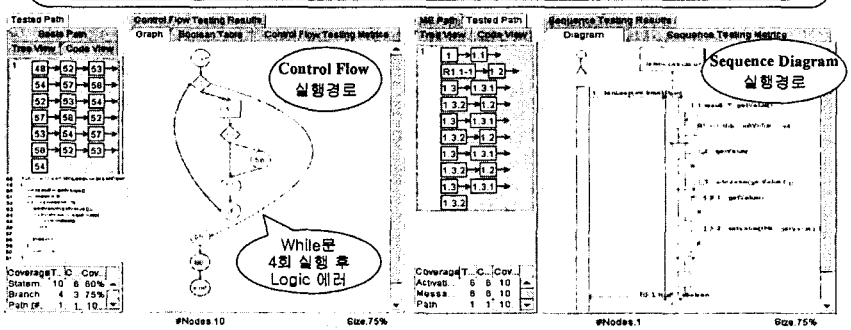
Pass/Fail/Error 분석 보고서: Test Data의 입력 값과 기대 값 평가
-각 Test Case의 실행 경로를 각 다이아그램/그래프와 연계하여 식별
-Run-time Error 분석: Error Trace, Error Message(소스 및 시나리오 에러 위치(라인) 등)



RESORT 테스트 기능(모니터링)

Testing – 모니터링 (단위/ 통합)

Pass/Fail/Error에 대한 실행 경로를 정밀 분석
-Unit Testing / Integration 실행 경로
-실행된 문장에 대한 실행 경로 하이라이팅 (source_code 와 연동)

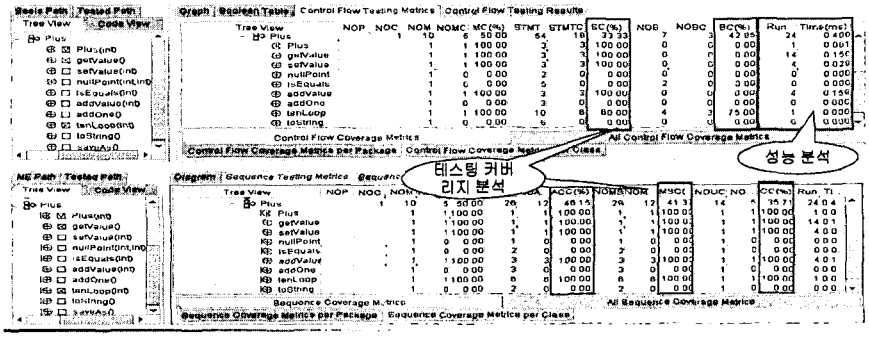


RESORT 테스트 기능(결과 분석)

Testing – 결과 분석 (단위/ 통합)

Coverage 분석
- -Control Flow Coverage: Statement, Branch
- - Data Flow Coverage: All-DU Path, All-C-Uses Path, All-P-Uses Path
- -Sequence Coverage: Activation, Message-send

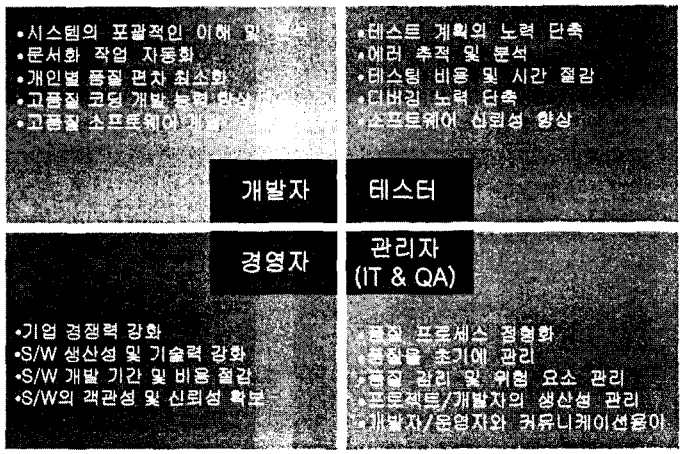
성능(Time/Run) 분석 : 성능 병목 현상 분석



Soft4-Soft

RESORT 기대 효과

■ 사용자의 기대 효과



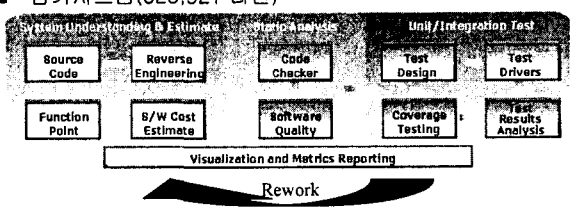
Soft4-Soft

Part 2: S/W 감리 적용 사례

- 소프트웨어 품질 감리
- 코드 품질 검사
- S/W 품질 측정
- Sample 품질 분석 보고서(Word)

소프트웨어 품질 감리

- 품질 감리 목적
  - S/W 품질 향상
  - S/W 성능 향상
- 소프트웨어 품질 측정 기법
  - 소스코드의 표준 검사
  - S/W 품질 측정
- 측정 대상 시스템
  - 운영시스템(397,384 라인)
  - 참가시스템(623,921 라인)



Soft4Soft

코드 품질 검사

- 목적
  - 프로젝트의 코딩 규칙 준수 여부
  - S/W 위험 요소의 초기 관리
  - 코드 품질 객관적 평가 및 관리
  - 성능 테스팅 비용 절감
  - 프로젝트의 프로그래밍 표준화를 위한 교육효과

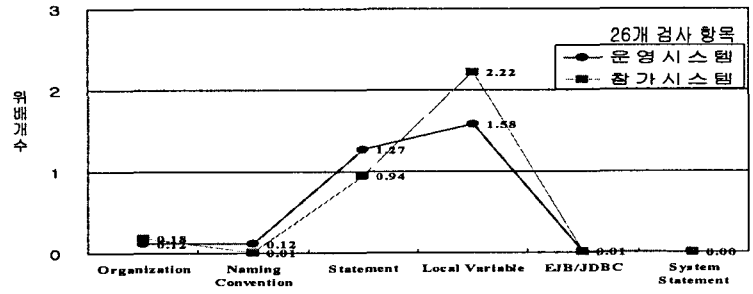
품질	품질 특성(측정 요소)	기대효과	측정 범위
코드 이해성 및 판독성 검사	문서 layout(문서화) -명명체계(Naming) -드러 쓰기,라인 각 -상수 사용 규칙	-판독성 향상 -이해성 향상 -유지보수 용이 -프로그램 개발 능력 향상	-File -Class -Method
코드 성능성 검사	성능 및 메모리 누수 -변수 사용 규칙 -제어문장 사용 규칙 -반복문 안에서의 사용 규칙 -EJB/JDBC 관련 리소스 해제여부 -Statements 사용 규칙 -System statements사용 규칙	-수행속도 증가 -메모리 누수 예방 -시스템 Testing 비용 절감 -프로그램 개발 능력 향상	-Method

코드 품질 검사

- 주요 코드 품질 점검 내용
  - 코드 이해성 및 판독성 검사
    - 주석, 이름 부여, "{ }", 라인 들여쓰기 및 길이 패턴 등
    - 중복 변수 및 하드 코딩 등
  - 코드 메모리 누수 검사
    - 사용하지 않는 변수 및 메소드
    - EJB 및 시스템 리소스 해제
  - 코드 성능성 검사
    - 반복문 내에서 메소드 호출 및 선언문
    - 스트링 연산 및 Console 메시지
    - 빈 Catch 구문(Application 에러 및 예외 처리)
    - Array 이름(Array Type 권장)
  - 프로젝트의 프로그래밍 표준화
    - 프로젝트 Framework 준수
    - 프로그램 구조 분석 : Return 및 Parameter의 수 등의 구조 품질
    - 코딩의 구조화 분석 : 최적화 코드 개발 유도

코드 품질 검사

- 500개 이상 위배한 Method 코드 규칙의 항목
  - 제어 구조문 괄호 규칙 (괄호 미사용) – 유지보수
  - Console 메시지 사용 제한 규칙– 실행 속도
  - 미사용 지역 변수 제한 규칙 – 메모리
  - 반복문내 메소드 및 선언문 제한 규칙 – 실행 속도
  - Array Type 사용 규칙 – 실행 속도
- 카테고리별(6종류) 코드 규칙을 위배한 각 Method 의 평균 개수



Soft4Soft

21

소프트웨어 품질 측정

- 목적
  - 품질 향상/개선을 위해 High-risk Class들을 진단
  - Project 관리 향상 – 품질 모니터링, 위험 관리, 개발자 및 품질관리자 대화 용이
  - 단위 및 통합 테스트 우선 순위 예측 – 테스팅 시간 및 노력 단축

품질	품질 특성(측정 요소)	기대 효과	측정 범위
S/W 고 위험성 측정	S/W 잠재적 에러 진단 -Class의 잠재적 에러 진단	-S/W 문제점 조기 발견 -S/W 위험 관리 -S/W 산출물간의 일치성 확인	-Class
S/W 최적화 측정	S/W 비최적화 진단 - Class 프로그램의 구조화 여부	-비효율적인 코드 제거 -프로그램 크기 축소 -프로그램 실행 시간 단축 -Testing 비용 절감	-Class
S/W 복잡성	S/W 품질 복잡도 진단 -Method의 크기 및 구조 품질 -Class의 크기 및 구조 품질 -Class의 OO 품질	-S/W 품질 향상 -S/W 생산성 향상 -S/W 유지보수 용이 -S/W 재사용성 -Testing 비용 절감	-Class -Method

Soft4Soft

22

소프트웨어 품질 측정

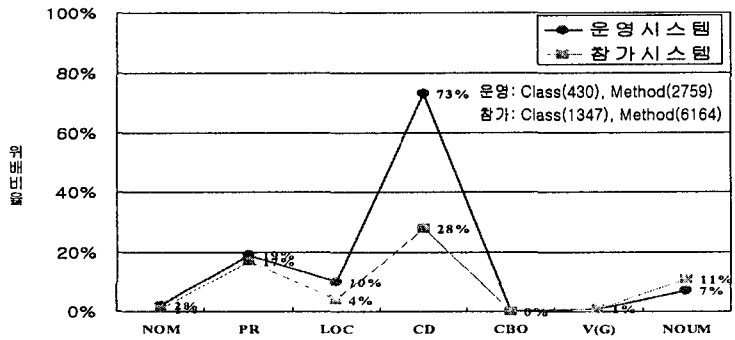
- 품질 측정 항목 (참가시스템: Class(1347), Method(6164))

품질 특성	품질 측정 요소	측정 값 (NASA 기준)	품질 평가 요소
S/W 고위험성	NOM(number of Methods)	0≤ NOM ≤62 (1≤ NOM ≤40)	-S/W 잠재적 에러 -S/W 구조 (설계 일치성)
S/W 최적화	PR(Purity Ratio)	0.13≤ PR ≤4.79 (1 ≤ PR)	-코드 최적화
S/W 복잡성	LOC(Lines of Code)	5≤LOC≤4949 (15≤ LOC ≤500)	-S/W 복잡도
	CD(Comment Density)	0%≤ CD ≤100% (20% ≤ CD)	-이해성
	CBO(Coupling Between Objects)	0≤ CBO ≤13 (0≤ CBO ≤5)	-모듈성(재사용성)
	V(G)(Cyclomatic Complexity)	1 ≤ V(G) ≤ 89 (1 ≤ V(G) ≤ 20)	-S/W 복잡도 -성능성 -테스팅 노력
	NOUM(Number of Used Methods)	1 ≤ NOUM ≤ 891 (1 ≤ NOUM ≤ 100)	-S/W 복잡도 -성능성 -테스팅 노력

Soft4Soft

소프트웨어 품질 측정

- 10% 이상 적합하지 못한 S/W 품질의 항목
  - 코드의 최적화 및 코드 주석 - 유지보수
  - 과다한 코드 라인 - S/W 복잡도
  - 과다한 메소드 실행(호출) 수 - 성능 저하, 테스팅 및 유지보수 어려움
- S/W 품질 요소(7종류)별 권고 기준을 초과한 평균 비율



Soft4Soft


If you cannot MEASURE it, you cannot IMPROVE it
㈜소프트4소프트
대전광역시 유성구 문지동 103-6 ICU 창업보육센터 T215호
Tel : 042-866-6633~2
Fax: 042-866-6626
구매 및 데모 문의 : sales@soft4soft.com
기술 및 일반 문의 : info@soft4soft.com
www. soft4soft.com
Soft4Soft
25

Sample 시스템
품질평가보고서

-RESORT for Java-

(Report)

Version x.x

2004. 10. 12.

Soft4Soft Co.,Ltd.
(주) 소프트4소프트

목차

그림

1. 개요

1.1 목적

CBD 방법으로 개발 중에 있는 " Sample 시스템" 에 대한 품질 평가 보고서이다.

1.2 진단 대상

" Sample 시스템" 자바 코드(JSP .java 포함)

1.3 진단 기간

2004 년 08 월 31 일(화)

1.4 수행 기관 및 품질 분석 도구

- (주)소프트 4 소프트(품질 측정 및 진단)
- RESORT for Java Toolset

1.5 진단 방법

진단 방법은 [그림 1]의 RESORT for Java 품질 프로세스의 도구들 중에서 정적 분석 도구인 아래의 도구를 사용하여 Code 품질 및 S/W 품질을 진단한다. 단, 본 분석 보고서에서는 단위 및 통합 테스팅을 적용하지 않았다.

- Code Checker (coding rule checker)
- Software Metrics (software quality measure)

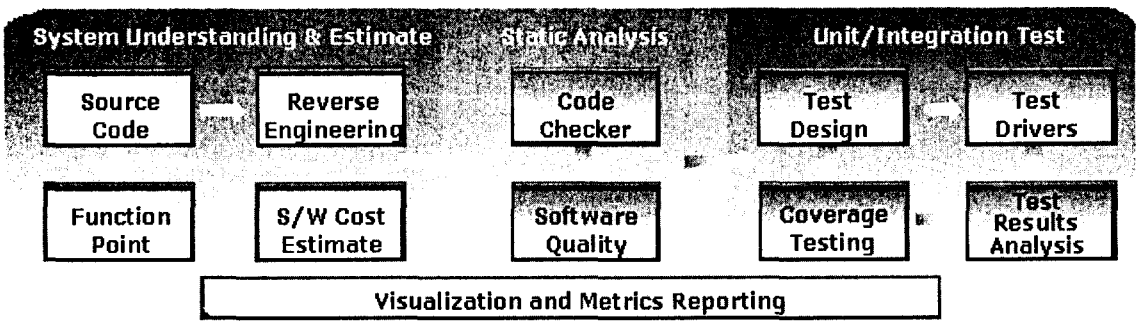


그림 1 RESORT for Java 품질 프로세스

1.6 코드 품질 진단 내용

코드 품질 진단은 이해성 및 판독성, 그리고 성능성의 품질 향상을 목적으로 소스 코드를 검사한다.

품질 특성	품질 부특성	기대 효과
코드 이해성 및 판독성 검사	문서 layout(문서화) -명명체계(Naming) -드려 쓰기,라인 폭 -상수 사용 규칙	-판독성 향상 -이해성 향상 -유지보수 용이 -프로그램 개발 능력 향상
코드 성능성 검사	성능 및 메모리 누수 -변수 사용 규칙 -제어문장 사용 규칙 -반복문 안에서의 사용 규칙 -JDBC 관련 리소스 해제여부 -Statements 사용 규칙 -System statements 사용 규칙	-수행속도 증가 -메모리 누수 예방 -시스템 Testing 비용절감 -프로그램 개발 능력 향상

1.7 S/W 품질 진단 내용

S/W 품질 진단은 고 위험성, 최적화 그리고 복잡성의 품질 향상을 목적으로 S/W 품질을 측정한다.

품질 특성	품질 부특성	기대 효과
S/W 고위험성	S/W 잠재적 에러 진단 -Class 의 잠재적 에러	-S/W 문제점 조기 발견 -S/W 위험 관리 -S/W 산출물간의 일치성 확인
S/W 최적화	S/W 비최적화 진단 -Class 프로그램의 구조화 여부	-비효율적인 코드 제거 -프로그램 크기 축소 -프로그램 실행 시간 단축 -Testing 비용 절감
S/W 복잡성	S/W 품질 복잡도 진단 -Method 의 크기 및 구조 품질 -Class 의 크기 및 구조 품질 -Class 의 OO 품질	-S/W 품질 향상 -S/W 생산성 향상 -S/W 유지보수 용이 -S/W 재사용성 -Testing 비용 절감

1.8 종합 검토 의견

코드 품질 점검 결과, 파일 및 클래스의 코드 품질은 매우 양호하나, 메소드의 코드 품질은 미사용 변수, 성능 저하 및 메모리 누수의 원인이 되는 규칙에서 상당히 많이 위배되어 관련 코드를 표준에 따라 수정 또는 삭제할 필요가 있다.

또한 S/W 품질 측정 결과, 향후 운영 및 유지보수성의 효율성을 높이기 위해서 코드의 최적화(구조화), 코드 주석, 그리고 과도한 메소드 실행(호출) 부분에 추가적인 개선활동이 필요하다.

마지막으로 6 개의 중복 클래스에 대한 재점검이 반드시 필요하다.

2. 시스템 구조 분석

2.1 시스템 통계

" Sample 시스템" 에서 정의된 Java의 구성 요소 통계는 아래 표와 같다.

Lines of Code	Package	File	Class (Inner)	Interface (Inner)	Method	Constructor	Variable
333,646	202	1,678	1,347(22)	393(40)	6,041	123	463

총 6개 클래스가 서로 다른 패키지에서 중복되어 존재하고 있다. 이에 대한 검토가 반드시 필요하다 [부록 1. 중복 Class 참조].

2.2 패키지 구조

" Sample 시스템" 은 아래 [그림 2]와 같이 크게 3 개의 큰 패키지로 구성되어 있으며, 총 패키지 수는 202 개 이다 (세부 패키지에 대해서는 기술하지 않는다).

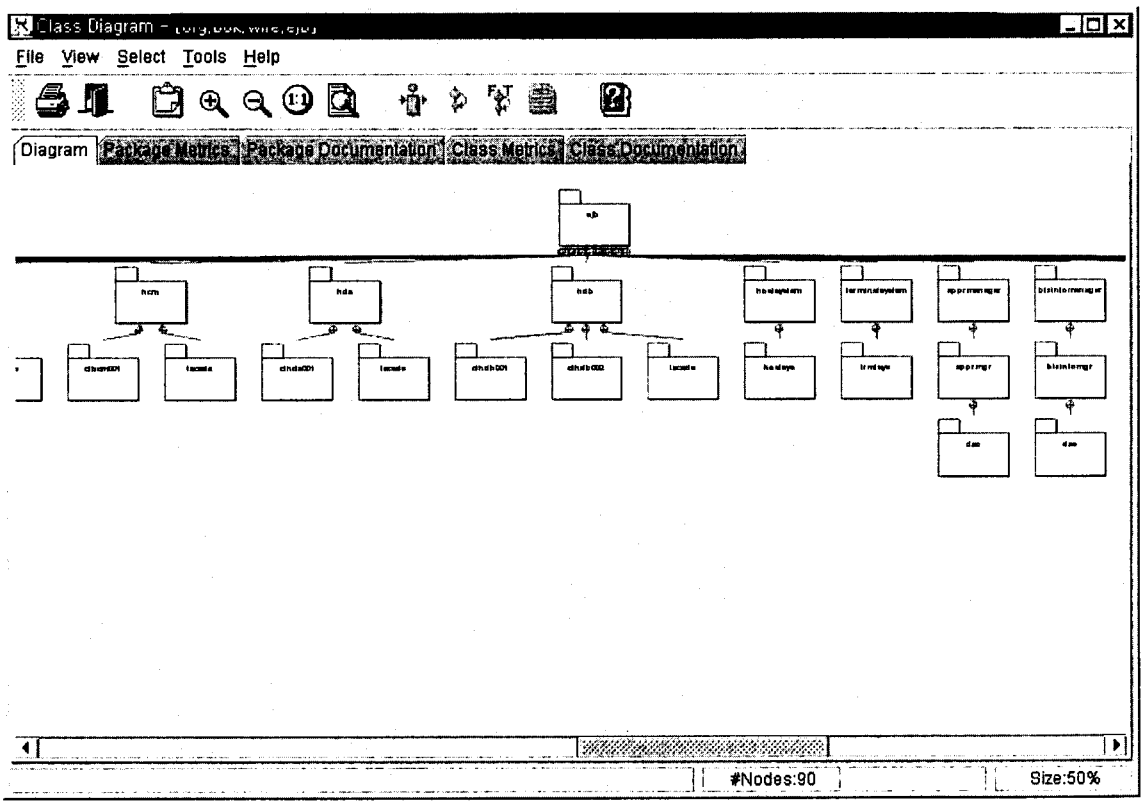


그림 2 패키지 다이아그램

2.3 클래스 구조

시스템의 총 클래스(내부 클래스 포함) 수는 1,347 개, 인터페이스(내부 인터페이스 포함) 수는 393 개로 구성되어 있다. 가장 많은 메소드 수를 포함하고 있는 RoleDAO 클래스에 대한 org.ejb.dao 패키지의 클래스 다이아그램은 [그림 3]과 같다. 이 클래스는 62 개의 메소드와 4,949 라인으로 구성 되어 있어, 이러한 클래스를 이해, 재사용 및 유지보수하기가 어려울 뿐만 아니라 클래스의 복잡도를 더 증가시킨다. 이 경우는 클래스를 많은 클래스의 구조로 분할해야 할 필요가 있다.

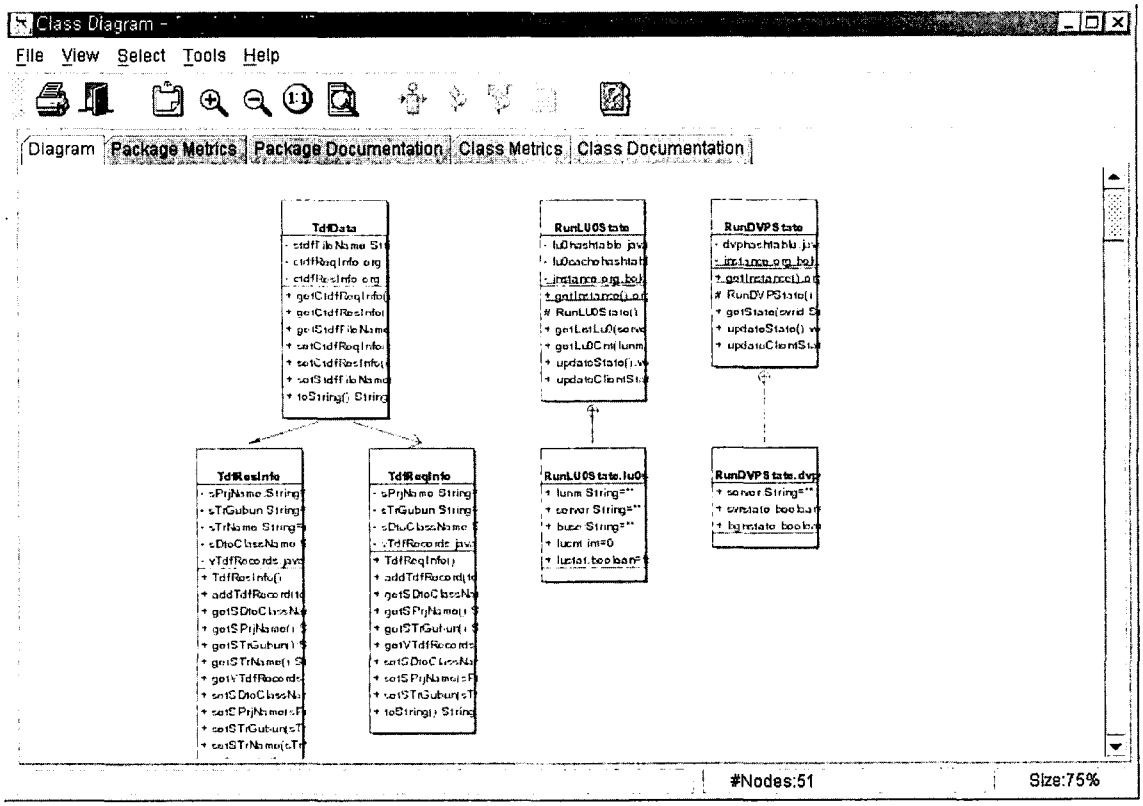


그림 3 org.ejb.dao 패키지의 클래스 다이아그램

2.4 메소드 구조

시스템의 총 메소드(생성자 포함) 수는 6,164 개 이며, 이 중에서 891 개의 메소드를 실행하고 있는 org.util 클래스의 run() 메소드의 시퀀스 다이아그램은 [그림 4]와 같다. 이 시퀀스 다이아그램에서 보는 바와 같이 이 메소드는 891 개의 메소드와 229 개의 객체를 사용하고 있어, 시스템의 성능을 저하 시킬 수 있으며, 또한 유지보수 및 이해의 어려움을 초래한다. 그러므로 메소드의 수를 줄일 필요가 있다.

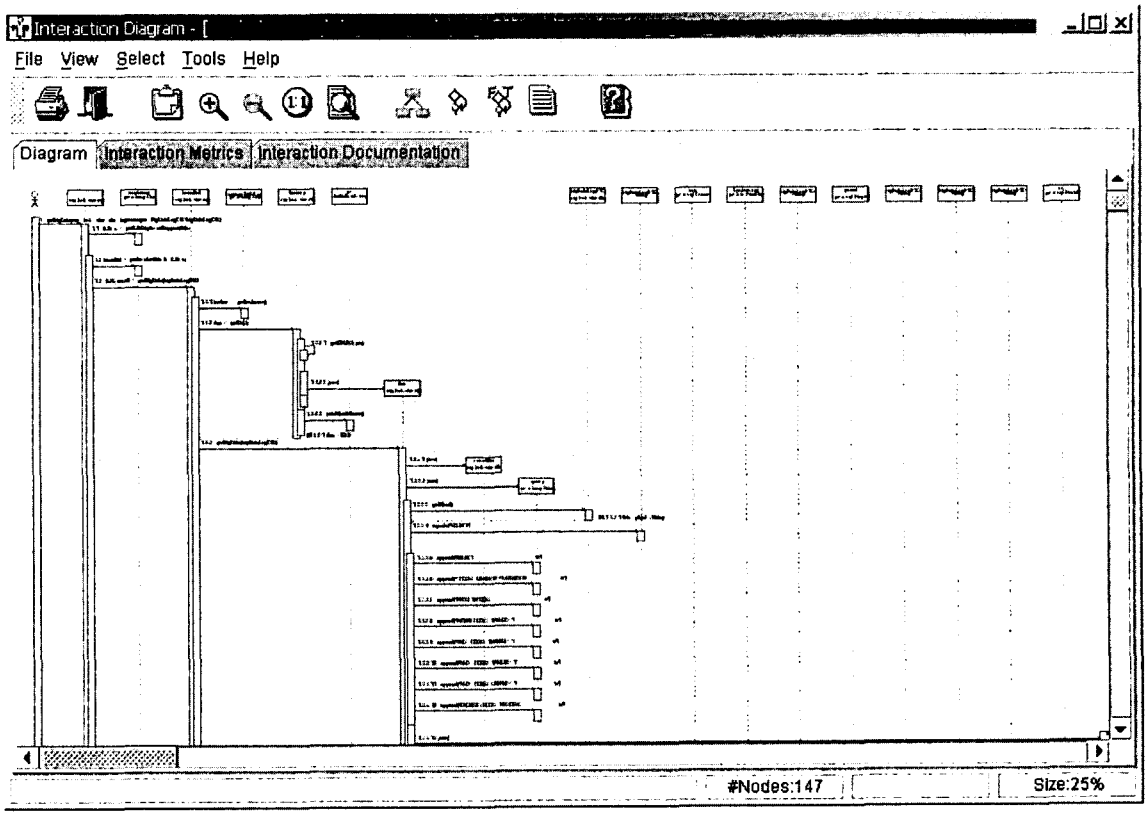


그림 4 run() 메소드의 시퀀스 다이아그램

3. 코드 품질 분석

코드 표준의 점검 사항을 지킴으로써, 소스코드에 대한 판독성 향상 및 유지보수 비용을 절감할 뿐만 아니라, 오류에 대한 사전 예방을 할 수 있는 이점이 있으며, 또한 제품을 배포시 패키징 하기가 쉬워 패키징 비용을 절감해 준다.

3.1 코드 품질 검사 결과

RESORT for Java 의 Code Checker Tool 의해 제공되는 각 단위별 코드 규칙 중에서 A 사의 프로그래밍 가이드(12 항목)와 RESORT 도구의 성능관련 주요 코드 규칙(24 항목) 등을 포함하여 총 36 개 항목의 코드 품질을 검사한다.

3.1.1 주요 검사 항목

다음은 코드 표준 점검 항목을 기술한 것이다.

Scope	Category	Item	측정 기준	A 사	Tool
File (4)	Organization (1)	File Size	파일당 2000 라인 이하	O	
		File Comment	파일 주석 점검		
		Class Declaration	파일당 1 클래스정의		
	Naming Convention (1)	File Name Length	이름 50 자 이내		
		Package Naming	소문자로 시작		O
		Import Declaration	" *" 사용 점검		
	Statement (2)	Line Length	한 라인 132 자 이내	O	
		Indentation	4 칸(들여쓰기)	O	
Class (6)	Organization (2)	Class Size	클래스당 500 라인이내		
		Class Comment	클래스 주석 점검		
		Class Declaration Brace	Brace 다음라인 시작		
		Class Modifier	사용불가 Modifier 점검		
		Failure Definition Variable	미사용 변수 점검		O
		Volatile Variables	Volatile 변수 점검		
		Default Constructor	디폴트 생성자 점검		
		Inheritance Thread	Thread 상속받는클래스 점검		O
	Naming Convention (3)	Class Naming	대문자로 시작	O	
		Class Name Length	이름 50 자 이내		
		Variable Naming	소문자로 시작	O	
		Constant Variable Naming	대문자로 시작	O	
		AMImpl Inheritance Naming	~ AMImpl 클래스점검		
		EOImpl Inheritance Naming	~EOImpl 클래스점검		
	Statement(1)	Synchronized Statement	Synchronized 키워드 점검		O

Scope	Category	Item	측정 기준	A 사	Tool
Method (26)	Organization (2)	Method Size	메소드당 50 라인이내		
		Method Comment	메소드 주석 점검		
		Method Declaration Brace	Brace 다음라인 시작		
		Method Modifier	사용불가 Modifier 점검		
		Parameter Counts	파라미터 5 개 이내		O
		Failure Definition Method	미사용 메소드 점검		O
		Throws clause for Method	Try 문장 점검		
	Naming Convention (1)	Method Naming	소문자로 시작	O	
		Method Name Length	이름 50 자 이내		
	Statement (11)	Control Structure Brace	Brace 붙여 쓰기 점검	O	
		Switch Statement	CASE 의 주석 또는 Br eak 점검	O	
		Throw Statement	If 의 Throw 점검		
		Return Statement	메소드별 1 개 이내		O
		Debug Statement	Console 메시지 점검		O
		Casting Statement	클래스의 형변환 점검		
		Additive Compound Assignment	" +=" 연산을 점검 (StringBuffer 사용)	O	
		Floating Point Values	부동소수점" ==" 연산을 점검		
		Hard Coding	하드 코딩된 문장점검		
		Empty Block Body	빈 제어 구조문 점검		O
		Empty catch/finally Block	빈 Catch 구문점검		O
		Non return in the finally Block	Finally 의 Return 문장 점검		O
		Nested try Statement	중첩 Try 문장 점검		O
		Nested Synchronized Statement	중첩 Synchronized 문장 점검		O
		Static Variable Access	정적 변수 접근식 점검		
		Used Static Variables	정적 변수문장 점검		
		Static Method Call	정적 메소드 호출 점검		
		Method Call in Loop Conditions	반복조건절의 메소드 호출 점검		O
	Local Variable (5)	Local Variable Naming	소문자로 시작	O	
		Failure Definitions	미사용 지역변수 점검		O
		Collection Declarations	Array Name 점검	O	
		Declaration statements in the Loops	반복문내의 선언문점검		O
		Duplicated Variable Names	전역 및 지역 변수의 동일이름 점검		O
	BC4J & JDBC (5)	getRowCount() Invocation	메소드 호출 점검		
		setWhereClause() Invocation	메소드 호출 점검		
		setQuery() Invocation	메소드 호출 점검		
		registerInfo() invocation	메소드 호출 점검		
		Class forName() Invocation	메소드 호출 점검		

		commit()-rollback() Invocation	Try-Catch 문의 메소드 호출 점검		O
		One try-catch per close() Invocation	하나의 Try 문에 Close( )메소드 호출 점검		O
		createRootApplicationModule() Invocation	메소드 호출 점검		
		commitChanges() and rollbackChanges() Invocation	메소드 호출 점검		
		Release JDBC	Close() 자원해제 점검		O
		Release JMS	JMS 자원 해제 점검		O
		Release Socket or I/O	I/O 자원 해제 점검		O
		Release PosBusinessLogicFactory	리소스 점검		
		Release Application Module	클래스 자원해제점검		
		Release Configuration	리소스 점검		
		Throw statement in the catch block	Persist()/start()메소드의 Try-catch 문 점검		
		Session Declaration	Session 선언문 점검		
		Review the runProgram()	메소드 호출 점검		
	System Statement (2)	System.gc() Invocation	메소드 호출 점검		O
		System.exit() Invocation	메소드 호출 점검		O
		Runtime.runFinalization() Invocation	메소드 호출 점검		
		Runtime.runFinalizersOnExit() Invocation	메소드 호출 점검		
		Runtime.exec() Invocation	메소드 호출 점검		
		Thread Methods Invocation	Thrasd 의 폐기된 메소드 점검		
		Don't use the ThreadGroup	ThreadGroup 사용점검		

3.1.2 파일별 코드 검사 결과

파일의 코드 규칙은 문서, 이름 부여 그리고 문장 규칙으로 구분할 수 있다.

파일의 코드 표준 점검 결과는 문서 규칙에서 약간 위배되었지만 이름 부여 규칙 및 문장 규칙에서는 위배 항목이 없다. 만약, 라인 길이 및 들여 쓰기의 문장 규칙에서 많이 검출되면, 코드 판독성의 어려움을 초래할 것이다 [부록 2. 파일코드검사 참조].

Scope	Category	Item	위배 개수	A사 지침
File (1678)	Organization	File Size	5	O
	Naming Convention	Package Naming	0	
	Statement	Line Length	0	O
		Indentation	0	O

Scope	Category	총 위배 개수	파일별 평균 위배 개수
File (1678)	Organization	5	0.002
	Naming Convention	0	0
	Statement	0	0

㈜ 파일별 평균 위배 개수 = 총 위배 개수 / 총 파일의 수

각 파일별 코드 품질에 대한 평균 위배 개수는 [그림 5]와 같다.

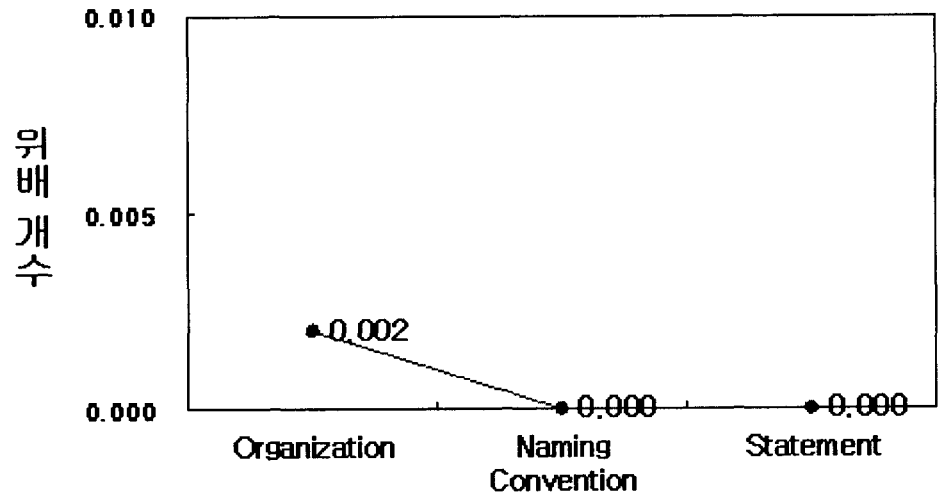


그림 5 각 파일별 코드 품질 평균 위배 개수

3.1.3 클래스(인터페이스)별 코드 검사 결과

클래스의 코드 규칙은 문서, 이름 부여 그리고 문장 규칙으로 구분할 수 있다.

클래스의 코드 표준 점검 결과는 문서 규칙(Thread 를 상속 받는 클래스 및 미사용 변수(메모리 누수 원인), 이름 부여 규칙(클래스, 변수 및 상수 변수 이름), 그리고 문장 규칙(자료 동기화 코드 규칙(자료 무결성 중시(속도 저하 요인)))에서 약간 검출되었다. 약간 위배된 이름 부여 규칙들은 수정되어야 하며, 향후 코드의 유지보수 및 이해성의 어려움을 초래할 것이다 [부록 3. 클래스코드검사 참조].

Scope	Category	Item	위배 개수	A 사 지침
Class&Interface (1740)	Organization	Failure Definition Variable	15	
		Inheritance Thread	11	
	Naming Convention	Class Naming	13	O
		Variable Naming	29	O
		Constant Variable Naming	4	O
	Statement	Synchronized Statement	49	

Scope	Category	총 위배 개수	클래스별 평균 위배 개수
Class&Interface (1740)	Organization	26	0.01
	Naming Convention	46	0.02
	Statement	49	0.02

㈜ 클래스별 평균 위배 개수 = 총 위배 개수 / 총 클래스의 수

각 클래스별 코드 품질에 대한 평균 위배 개수는 [그림 6]과 같다.

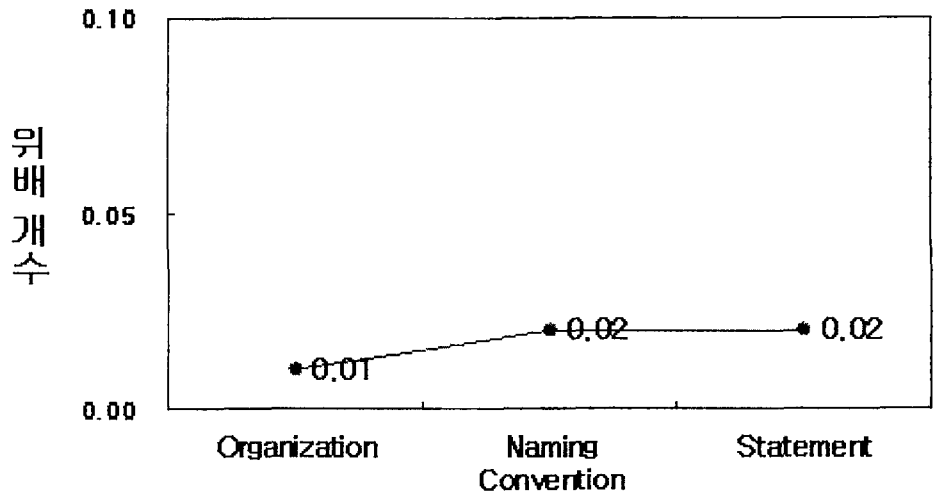


그림 6 각 클래스별 코드 품질 평균 위배 개수

3.1.4 메소드(생성자)별 코드 검사 결과

메소드의 코드 규칙은 문서, 이름 부여, 문장, 변수, JDBC 및 시스템 문장 사용 규칙으로 구분할 수 있다.

문서 부여 규칙에서는 많은 파라미터를 사용함으로써 간접적으로 메소드의 복잡성을 예측할 수 있고, 약간의 미사용 메소드들은 메모리 누수의 원인이 됨으로 삭제되어야 한다.

이름 부여 규칙에서는 약간 위배된 메소드 및 변수 이름들을 수정하야 하며, 향후 코드의 유지보수 및 이해성의 어려움을 초래할 것이다.

문장 사용 규칙에서는 많은 제어문의 괄호 규칙을 위배함으로써 코드 판독성의 어려움을 초래하게 되며, 그리고 두개 이상의 Return 사용을 사용함으로써 간접적으로 메소드의 복잡성을 예측할 수 있다. 과다한 Console 메시지 사용, 빈 Catch 구문(Application 에러 및 예외 처리), 그리고 반복문에서 메소드 호출들은 실행 속도의 저하시키는 원인이 됨으로 수정되어야 한다.

변수 사용 규칙에서는 중복 변수를 사용함으로써 향후 유지보수의 어려움을 초래하게 된다. 과다한 Array Name 사용(Array Type 사용 권장) 및 반복문에서 선언문 사용들은 실행 속도의 저하시키는 원인이 됨으로 수정되어야 한다. 마지막으로 아주 많은 미사용 변수들은 메모리 누수의 원인이 됨으로 삭제되어야 한다.

특히, 메소드 관련 코드는 시스템 운영시 직접적인 문제를 유발할 수 있기 때문에 반드시 위배된 실행 및 메모리 규칙은 반드시 수정/삭제하여야 한다 [부록 4 메소드코드검사 참조].

Scope	Category	Item	위배 개수	A사 지침
Method & Constructor (6164)	Organization	Parameter Counts	1065	
		Failure Definition Method	21	
	Naming Convention	Method Naming	35	O
	Statement	Control Structure Brace	3883	O
		Switch Statement	36	O
		Return Statement	235	
		Debug Statement	544	
		Additive Compound Assignment	10	O
		Empty Block Body	13	
		Empty catch/finally Block	253	
		Non return in the finally Block	0	
		Nested try Statement	30	
		Nested Synchronized Statement	0	
		Method Call in Loop Conditions	808	
	Local Variable	Local Variable Naming	7	O
		Failure Definitions	10753	
		Collection Declarations	1172	O
		Declaration Statements in the Loops	1678	

	JDBC	One try-catch per close()Invocation	0	
		commit()-rollback() Invocation	3	
		Release JDBC	0	
		Release JMS	0	
		Release Socket or I/O	0	
	System Statement	System.gc() Invocation	0	
		System.exit() Invocation	0	

Scope	Category	총 위배 개수	메소드별 평균 위배 개수
Method & Constructor (6164)	Organization	1,086	0.176
	Naming Convention	49	0.005
	Statement	5,812	0.942
	Local Variable	13,685	2.220
	JDBC	3	0.0004
	System Statement	0	0

㈜ 메소드별 평균 위배 개수 = 총 위배 개수 / 총 메소드의 수

각 메소드별 코드 품질에 대한 평균 위배 개수는 [그림 7]과 같다.

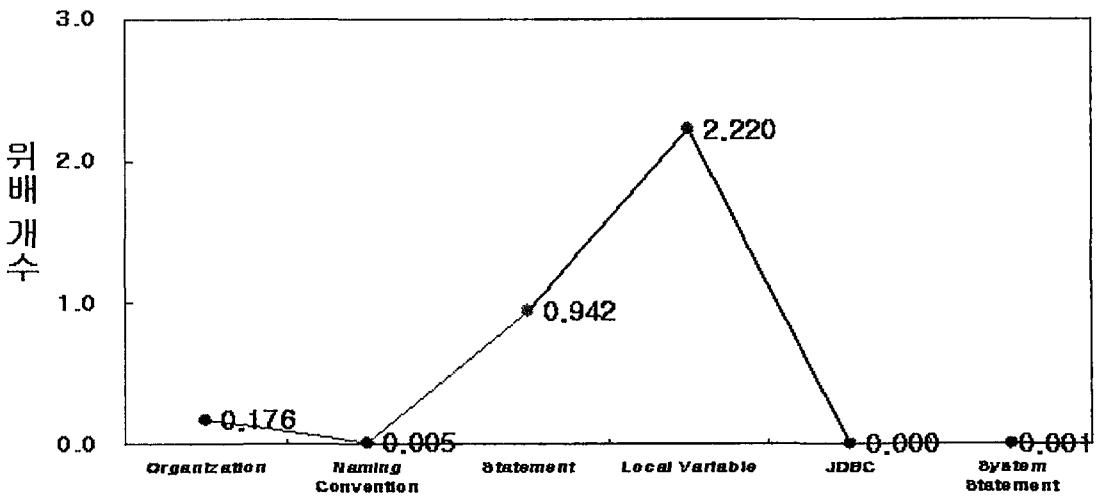


그림 7 각 메소드별 코드 품질 평균 위배 개수

3.2 도구와 A 사의 지침에 의한 코드 품질 종합 평가

파일 및 클래스의 코드 품질은 매우 양호하나, 메소드의 코드 품질은 미사용 변수, 성능 저하 및 메모리 누수의 원인이 되는 규칙에서 상당히 많이 위배되어 관련 코드를 표준에 따라 수정 또는 삭제할 필요가 있다. 특히, 500 개 이상 검출된 항목을 보면 다음과 같다.

(1) 메소드에서 제어 구조문의 괄호 붙여쓰기 규칙 - A 사 프로그래밍가이드
(2) 메소드에서 6 개 이상의 Parameter 사용 제한 규칙
(3) 메소드에서 Console 메시지 사용 금지 규칙
(4) 메소드에서 반복문내 메소드 호출 사용 금지 규칙
(5) 메소드에서 미사용 변수 삭제 규칙
(6) 메소드에서 Array Type 사용 규칙- A 사 프로그래밍가이드
(7) 메소드에서 반복문내 선언문 사용 금지 규칙

종합적인 코드 평가 결과로써, 시스템의 각 구성 요소에 대한 각 메소드/클래스/파일별 코드 품질의 평균 위배 개수는 [그림 8]과 같다.

Category(총 개수)	위배 개수	평균 위배 개수(위배개수/총 개수)
File (1678)	5	0.002
Class & Interface (1740)	121	0.695
Method & Constructor (6164)	20,635	3.347

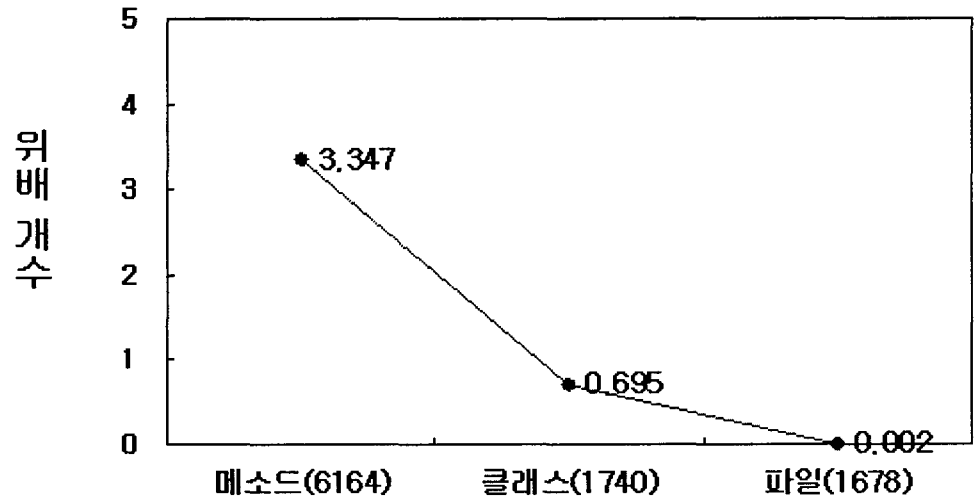


그림 8 각 메소드/클래스/파일별 코드 품질 평균 위배 개수

4. S/W 품질 분석

소프트웨어 품질은 코드의 최적화, 크기, 구조, 복잡도, 객체지향 특성 등의 품질 측정을 통해 소프트웨어 프로젝트의 관리, 제어, 모니터링을 함으로써 고품질의 S/W 개발 또는 S/W 유지보수의 비용을 줄일 수 있다. 또한 S/W 생산성을 높일 수 있을 뿐 아니라 S/W 신뢰성을 향상시킬 수 있다.

RESORT for Java 의 S/W Metrics Tool 의해 제공되는 품질 측정 요소 중에서 주요한 총 7 개 요소의 S/W 품질을 평가하며, 그 평가 목적은 아래와 같다.

품질 특성	품질 측정 요소	측정 대상	품질 평가 요소
S/W 고위험성	NOM(number of Methods)	Class	-S/W 잠재적 에러 -S/W 구조 (설계 일치성)
S/W 최적화	PR(Purity Ratio)	Class	-코드 최적화
S/W 복잡성	LOC(Lines of Code)	Class	-S/W 복잡도
	CD(Comment Density)	Class	-이해성
	CBO(Coupling Between Objects)	Class	-모듈성(재사용성)
	V(G)(Cyclomatic Complexity)	Method	-S/W 복잡도 -성능성 -테스팅 노력
	NOUM(Number of Used Methods)	Method	-S/W 복잡도 -성능성 -테스팅 노력

4.1 S/W 고위험성 품질 분석

클래스의 잠재적 에러 구조를 측정한다. 클래스의 잠재적 에러 구조는 1 차적으로 클래스 크기를 나타내는 NOM(Number of Methods)에 의해 결정된다. 메소드가 없는 클래스는 설계 문서와 코드간의 산출물 일치성 확인을 위한 재검토가 필요하다.

4.1.1 NOM(Number of Methods)

(정의) 이 메트릭스는 클래스 내부에 정의된 메소드 수를 측정하는 요소이다.

(분석) 클래스가 많은 메소드들을 가질수록 그 클래스에 대한 이해, 재사용 및 유지보수하기가 어려울 뿐만 아니라 클래스의 복잡도를 더 증가시킨다. 이 경우는 클래스를 두개 이상으로 분할해야 할 필요가 있음으로 설계의 재검토가 필요하다. 반면, 측정 값이 " 0" 이면, 사실 클래스가 아니라 단지 자료 구조만을 정의한 클래스이다.

(진단) 클래스의 잠재적 에러는 없으나(자료 구조임), 약 1%의 클래스가 과다한 메소드를 정의하고 있어 부적합하다(NASA 기준) [부록 5. NOM 참조].

Class	권고 기준 값	측정 값	최소값 초과 개수	최대값 초과 개수
1347	1≤ NOM ≤40	0≤ NOM ≤62	4	12

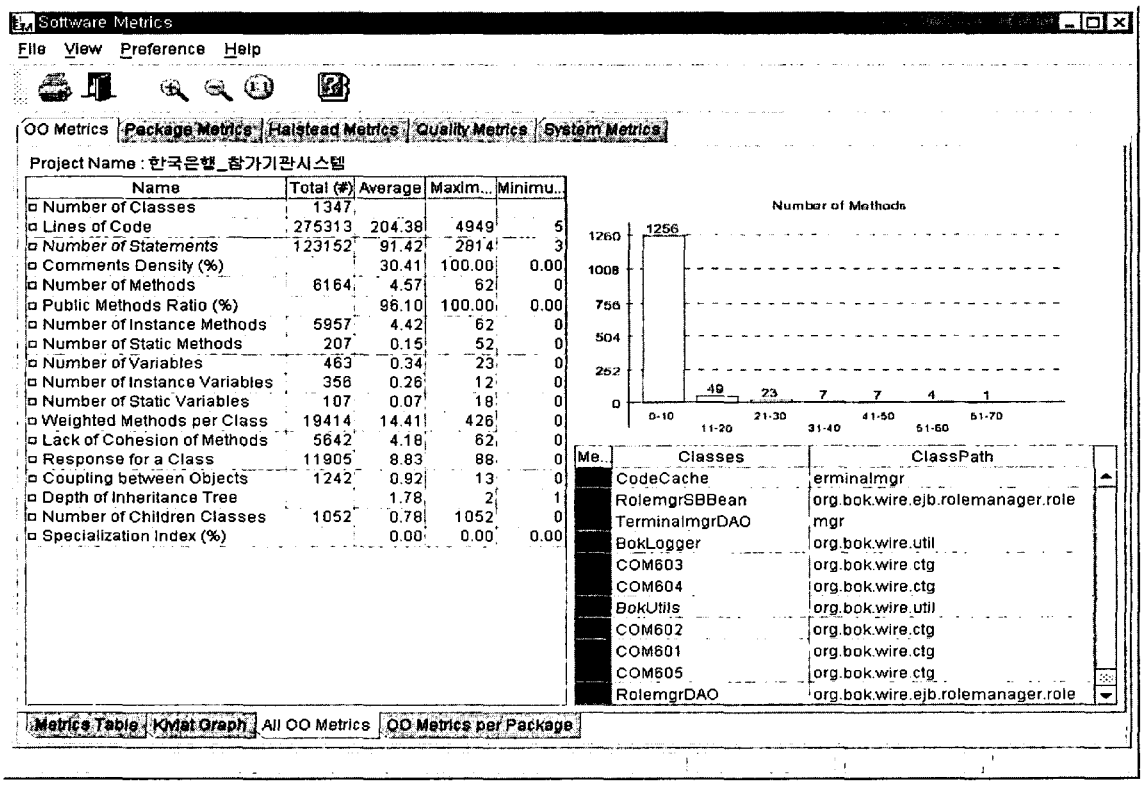


그림 9 NOM(Number of Methods) 메트릭스

4.2 S/W 최적화 품질 분석

클래스 코드가 잘 구조화된 코드인지를 측정한다. 최적화 되지 않은 클래스의 코드는 될 수 있는 한 재코딩할 필요가 있다.

4.2.1 PR(Purity Ratio)

(정의) 이 메트릭스는 클래스 코드가 최적화 여부를 측정하는 요소이다.

(분석) " PR = 1" 을 기준으로 1 보다 작으면 비 효율적인 코드(동일 변수 및 동일 부분식 중복 사용, 연속적인 연산자 사용)로써 판독성 및 유지보수성에 어려움이 있다. 1 보다 클수록 잘 구조화된 코드라 할 수 있다. 특히, 측정 값이 1 보다 작으면, 중복 또는 반복 코드가 있다는 것으로써 재코딩이 필요하다.

(진단) 약 17%의 클래스가 부적합하다 [부록 6. PR 참조].

Class & interface	권고 기준 값	측정 값	최소값 초과 개수
1740	1 ≤ PR	0.13≤ PR ≤4.79	305

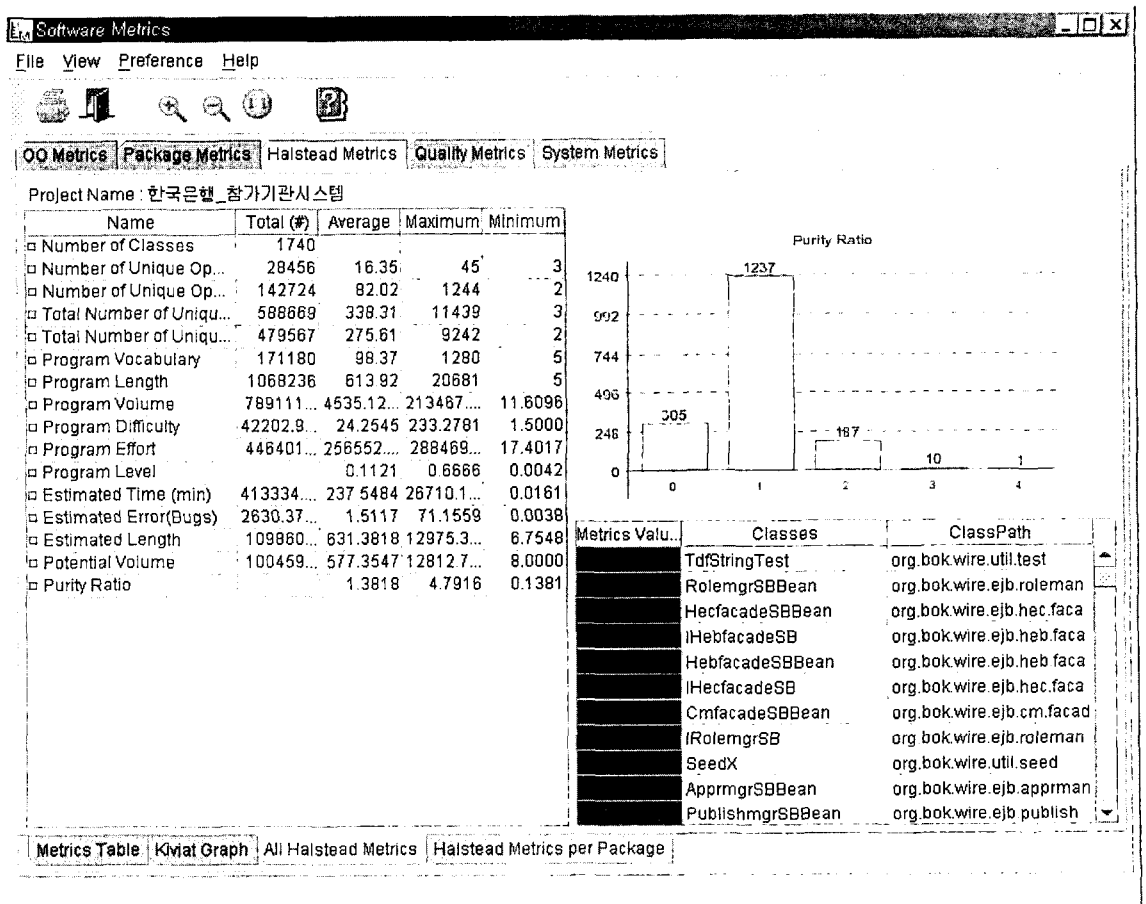


그림 10 PR(Purity Ratio) 메트릭스

4.3 S/W 복잡성 품질 분석

클래스 또는 메소드의 내부 속성인 크기 및 구조를 측정한다. 시스템 성능 및 테스팅 노력, 그리고 OO 재사용에 관련된 품질을 측정한다. 복잡한 구조 및 크기는 설계 또는 코딩의 재검토가 필요하다.

4.3.1 LOC(Lines of Code)

(정의) 이 메트릭스는 클래스의 물리적 코드 길이를 측정하는 요소로써 S/W 의 복잡도의 간접 측정을 나타낸다.

(분석) 클래스 코드 길이가 작게 유지되면 될수록 코드의 이해성 및 유지보수성이 향상된다. 반면, 과다한 코드 길이는 코드의 이해 및 유지보수를 어렵게 할 것이다. 그러므로 코드 길이를 제한하는 것이 바람직하다. 특히, 권고 기준 값을 초과한 클래스는 그 클래스를 유사 클래스와 통합하거나 두개 이상으로 분할해야 할 필요가 있음으로 설계의 재검토가 필요하다.

(진단) 약 4%의 클래스가 부적합하다(NASA 기준) [부록 7. LOC 참조].

Class	권고 기준 값	측정 값	최소값 초과 개수	최대값 초과 개수
1347	15≤ LOC ≤500	5≤LOC≤4949	10	45

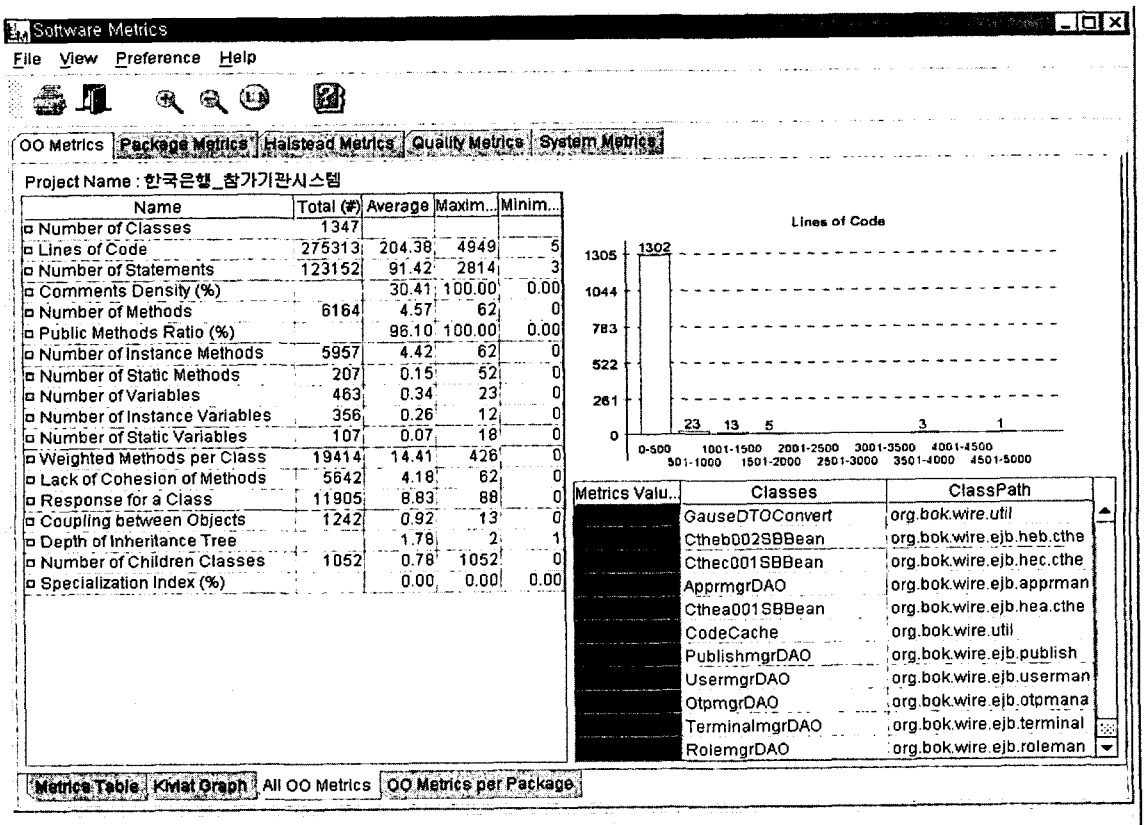


그림 11 LOC(Lines of Code) 메트릭스

4.3.2 CD(Comment Density)

(정의) 이 메트릭스는 클래스의 논리적 코드 길이에(Number of Statements, STMT) 대한 주석 블록의 비율을 측정하는 요소로써 코드 이해의 용이성을 측정한다.

(분석) 낮은 클래스의 주석 비율은 코드의 문서가 잘되 않았다는 것으로써 S/W 개발자나 유지보수자가 코드를 이해하는데 어렵게 한다. 특히, 권고 기준 값을 초과한 클래스는 주석 비율을 높일 필요가 있다.

(진단) 약 28%의 클래스가 부적합하다(NASA 기준) [부록 8. CD 참조].

Class	권고 기준 값	측정 값	최소값 초과 개수
1347	20% ≤ CD	0%≤ CD ≤100%	382

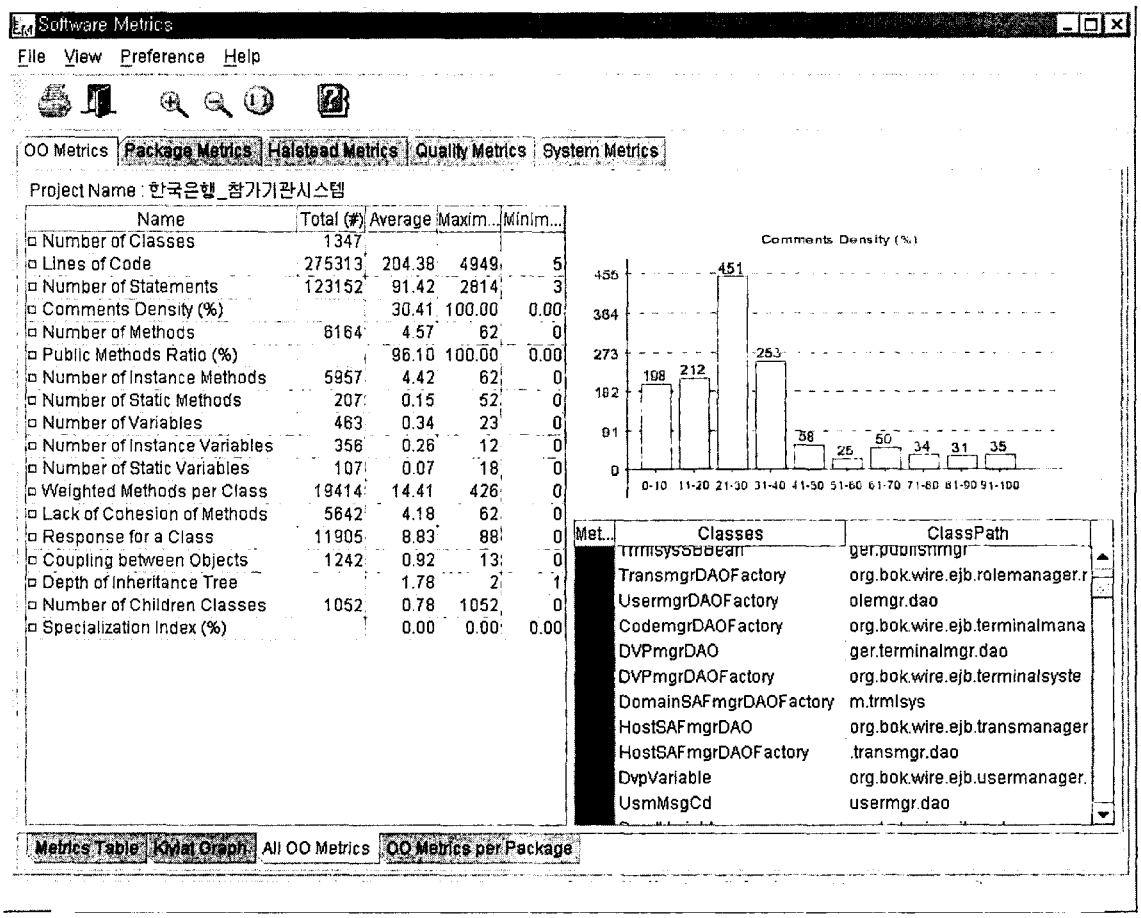


그림 12 CD(Comment Density) 메트릭스

4.3.3 CBO(Coupling Between Objects)

(정의) 이 메트릭스는 클래스들 간의 결합 정도를 측정하는 요소이며, 클래스의 재사용성과 관련이 있다.

(분석) 클래스 결합도가 최소로 유지되면 될수록 모듈성이 향상되며, 또한 재사용하기가 더 쉽다. 반면, 클래스의 과다한 결합도는 유지 보수 및 재사용 하기가 더 어려울 뿐만 아니라 테스팅의 복잡도를 초래하여 좀 더 엄격한 테스팅 노력이 요구 되어지기 때문에 바람직하지 않다. 특히, 측정 값이 " 0" 이면, 프로젝트내의 어떠한 클래스와도 관계가 없다는 것으로써 프로젝트의 구성요소가 아닐 수도 있다.

(진단) 약 0.1%의 클래스가 부적합하다(NASA 기준) [부록 9. CBO 참조].

Class	권고 기준 값	측정 값	최대값 초과 개수
1347	0≤ CBO ≤5	0≤ CBO ≤13	2

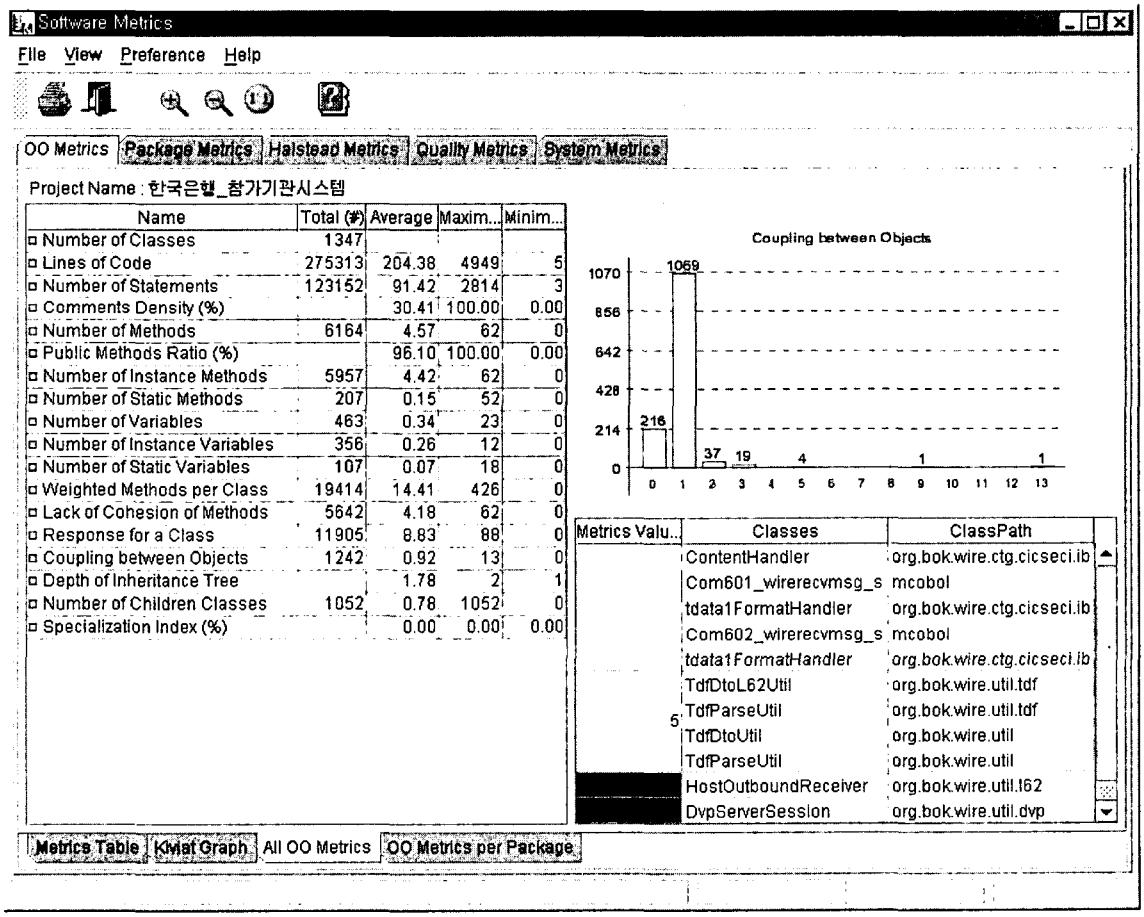


그림 13 CBO(Coupling Between Objects) 메트릭스

4.3.4 V(G)(Cyclomatic Complexity)

(정의) 이 메트릭스는 메소드의 알고리즘(제어 흐름) 복잡도를 측정하는 요소이며, 테스팅의 노력과 시스템 성능성에 관련이 있다.

(분석) 과다한 복잡도는 코드를 이해하기가 어려울 뿐만 아니라 복잡하고 위험한 프로그램이 될 수 있으며, 또한 단위 테스팅시 테스트 케이스(Test Case) 설계의 과다 비용을 초래한다. 이 경우는 과다한 제어 구조(Control Structure)를 포함하기 때문에 그 메소드를 두개 이상으로 분할해야 할 필요가 있음으로 설계의 재검토가 필요하다. 특히, 측정 값이 " 1" 이면, 기능의 결여일 수도 있다.

(진단) 약 1%의 메소드가 부적합하다(NASA 기준) [부록 10. V(G) 참조].

Method	권고 기준 값	측정 값	최대값 초과 개수
6164	1 ≤ V(G) ≤ 20	1 ≤ V(G) ≤ 89	40

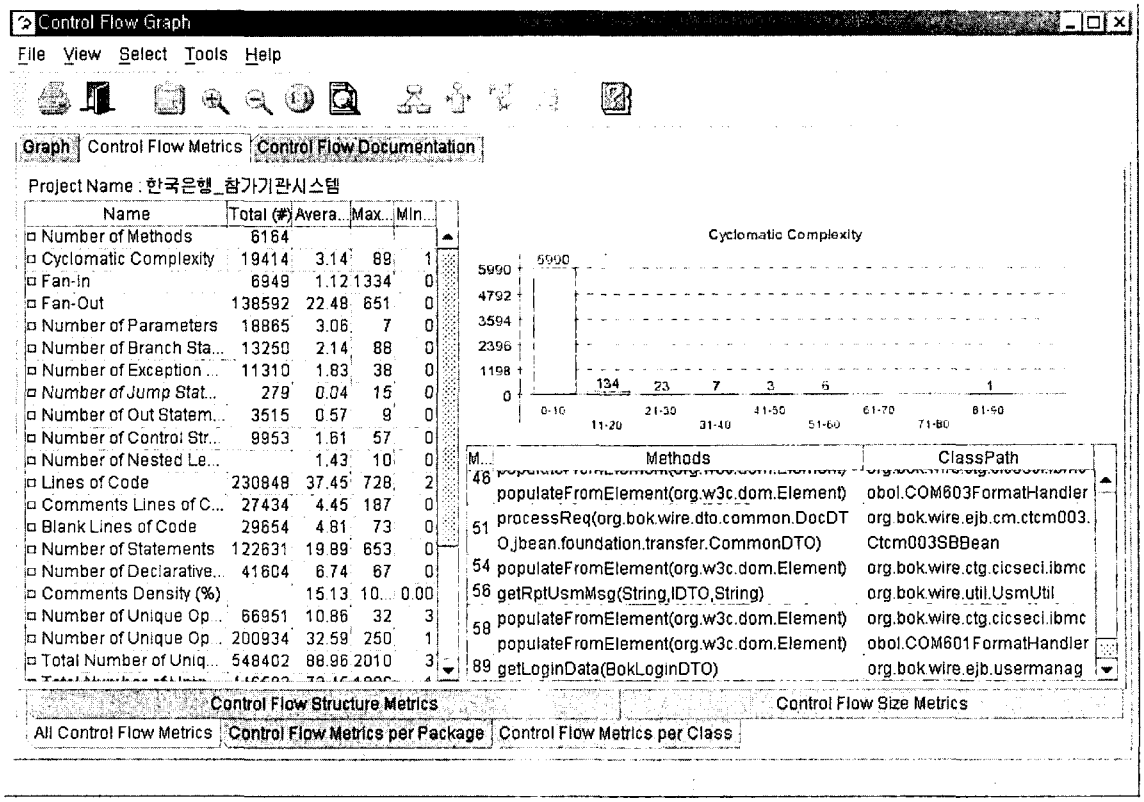


그림 14 V(G)(Cyclomatic Complexity) 메트릭스

4.3.5 NOUM(Number of Used Methods)

(정의) 이 메트릭스는 실행할 수 있는 메소드의 수를 측정하는 요소이며, 시스템 성능성과 관련이 있다.

(분석) 과다한 호출은 매우 복잡하고, 디버깅의 어려움 및 통합 테스팅의 노력을 증가시키는 아주 위험한 메소드이며, 특히 유지보수 및 이해의 어려움이 있다. 특히, 권고 기준 값을 초과한 메소드는 설계의 재검토가 필요하다.

(진단) 약 11%의 메소드가 부적합하다(NASA 기준) [부록 11 NOUM 참조].

Methods	권고 기준 값	측정 값	최대값 초과 개수
6164	1 ≤ NOUM ≤ 100	1 ≤ NOUM ≤ 891	688

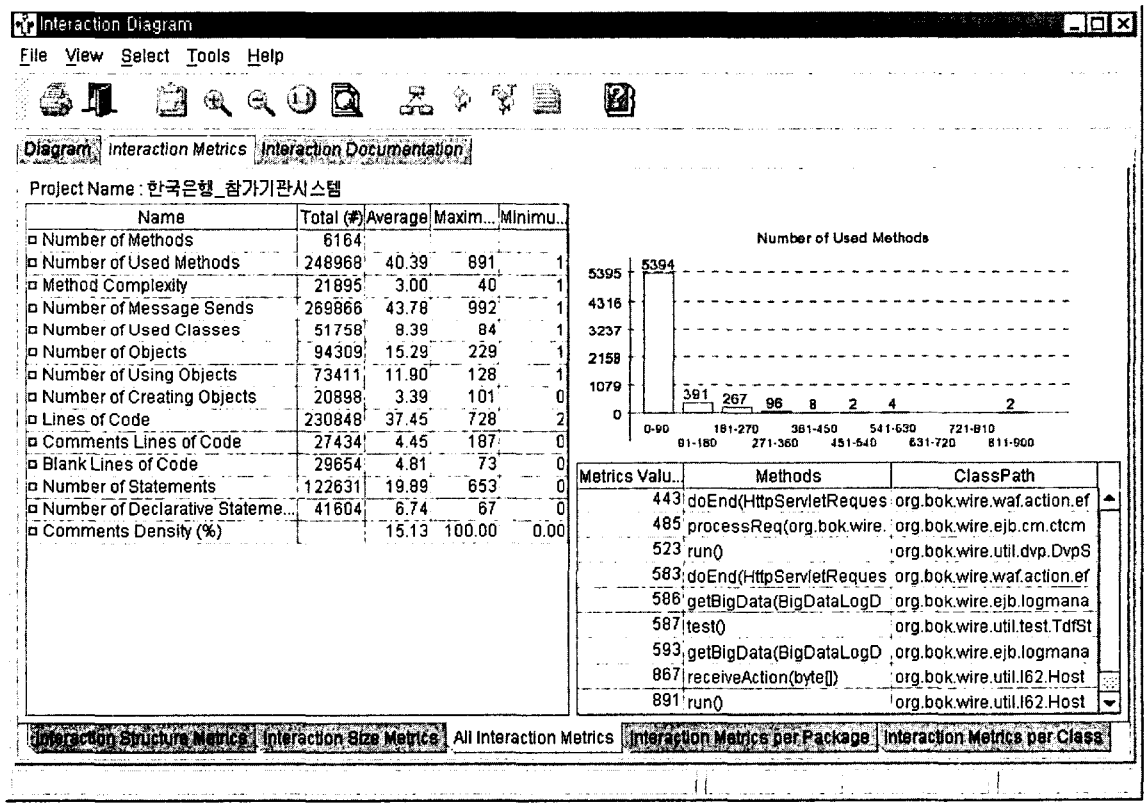


그림 15 NOUM(Number of Used Methods) 메트릭스

4.4 S/W 품질 종합 평가

S/W 품질 평가는 아래의 표와 같이 양호하나(권고 기준 초과 비율 10% 이하), 코드의 최적화(구조화), 코드 주석, 과도한 메소드 실행(호출) 부분에서는 추가적인 개선활동이 필요하다.

품질 특성	측정 범위	품질 측정 요소	권고기준 초과비율(%)
S/W 고위험성	-Class	-NOM(Number of Methods)	1%
S/W 최적화	-Class	-PR(Purity Ratio)	17%
S/W 복잡성	-Class	-LOC(Lines Of Code) -CD(Comment Density) -CBO(Coupling Between Objects)	4% 28% 0.1%
	-Method	-V(G))(Cyclomatic Complexity) -NOUM(Number of Used Methods)	1% 11%

유지보수관점에서는 80 메소드 이상(권고기준 2 배 이상)을 포함하는 클래스들은 없다. 만약, 있다면, S/W 구조와 크기가 매우 복잡하여 설계의 재검토가 필요하며, 향후 테스팅 및 유지보수 어려움의 잠재성들이 발생될 것이다 (부록 5 NOM 참조).

Class	ClassPath	클래스당 메소드수
없음		

성능관점에서는 다음과 같이 200 개 이상 메소드를 실행하는(권고기준 2 배 이상) 276 개의 메소드들은 성능 저하 요인, 테스팅 어려움, 그리고 유지보수 어려움을 초래할 수 있으므로 재점검이 필요하다 (부록 11 NOUM 참조, 본 고에서는 400 개 이상의 메소드 호출만 서술함).

Method	ClassPath	메소드 실행 수
doEnd	org.bok.wire.waf.action.efa.EFA5010HreAction	437
Generate	org.bok.wire.util.GenJournalLog	438
doEnd	org.bok.wire.waf.action.efb.EFB831HiqAction	443
processReq	org.bok.wire.ejb.cm.ctcm003.Ctcm003SBBean	485
run	org.bok.wire.util.dvp.DvpServerSession	523
doEnd	org.bok.wire.waf.action.efb.EFB837HiqAction	583
getBigData	org.bok.wire.ejb.logmanager.logmgr.dao.LogmgrDAO	586
test	org.bok.wire.util.test.TdfStringTest	587
getBigData	org.bok.wire.ejb.logmanager.logmgr.LogmgrSBBean	593
receiveAction	org.bok.wire.util.I62.HostOutboundReceiver	867
run	org.bok.wire.util.I62.HostOutboundReceiver	891